Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-3641 - Lenguajes de Programación I
Enero-Marzo 2015

Carnet:_____

Nombre:_____

Examen I

(25 puntos)

Pregunta 1	Pregunta 2	Pregunta 3	**Total**
12 puntos	6 puntos	7 puntos	**25 puntos**

Pregunta 1 - 12 puntos

Esta pregunta consta de seis (6) subpreguntas de selección, numeradas de 1.1 a 1.6. Cada una de las subpreguntas viene acompañada de cuatro posibles respuestas (a, b, c, d), entre las cuales sólo **una** es correcta. Ud. deberá marcar completamente y sin posibilidad de confusión la opción que considere correcta. Cada una de las subpreguntas tiene un valor de dos (2) puntos. Tres subpreguntas incorrectas eliminan una correcta.

1.1. El diseño de un lenguaje de programación establece que en todas sus implantaciones los números enteros deben tener 64 bits de precisión y su aritmética debe efectuarse utilizando complemento a dos. Tal decisión de diseño puede favorecer o no las siguientes características del lenguaje:

I. La portabilidad de los programas escritos en el lenguaje.
II. La portabilidad de la implantación del lenguaje.
III. La eficiencia de mantenimiento de programas escritos en el lenguaje.

¿Cuáles favorece?

a) I y II.
b) I y III.
c) II y III
d) Todas.

1.2. Un lenguaje interpretado, con alcance estático y funciones de primera clase **siempre** necesita:

a) Una tabla de símbolos del estilo Lista de Asociación.
b) Promover objetos al *heap* para toda clausura.
c) Diferir todas las verificaciones semánticas hasta la ejecución del programa.
d) Ninguna de las anteriores.

1.3. Una variable local a un procedimiento en un lenguaje compilado **siempre**:

a) Puede tener tiempo de vida más largo que tiempo de alcance.
b) Puede promoverse al *heap* para construir una clausura.
c) Sólo es accesible para el mismo procedimiento actualmente en ejecución, pero no para procedimientos llamados recursivamente.
d) Ninguna de las anteriores.

1.4. Considere un lenguaje imperativo, de propósito general, con alcance dinámico y sin clausuras. ¿Cuál de las siguientes verificaciones puede completarse enteramente de forma estática?

a) Verificar que una expresión produzca división entre cero.
b) Verificar que un subíndice no exceda los límites de un arreglo.
c) Verificar que toda variable recibe valores cónsonos con su tipo.
d) Verificar que toda excepción pueda ser atrapada.

1.5. La diferencia entre un módulo como administrador y un módulo como tipo radica en que:

a) El espacio de nombres del primero sólo contiene procedimientos, mientras que el espacio de nombres del segundo contiene tanto procedimientos como tipos de datos.

b) El primero sólo puede tener espacio de nombres cerrado, mientras que el segundo puede tener espacio de nombres abierto o cerrado.

c) En el primero los datos se pasan a las funciones como argumentos explícitos, mientras que en el segundo se pasan implícitamente.

d) El primero sólo es aplicable en un lenguaje imperativo o funcional, mientras que el segundo sólo es aplicable en un lenguaje orientado a objetos.

1.6. El ambiente de ejecución de un lenguaje con almacenamiento en *heap* requiere un manejador de memoria para administrar el espacio adicional. Considere las siguientes afirmaciones en relación a cualquier manejador de memoria:

I. *Best Fit* siempre es mejor que *First Fit* para controlar la fragmentación externa.

II. La fragmentación interna depende de la selección de tamaños para los bloques.

III. La variante Fibonacci para el "Buddy System" empeora la fragmentación interna.

¿Cuáles afirmaciones son *falsas*?

a) I y II solamente.

b) I y III solamente.

c) II y III solamente.

d) Todas son falsas.

Pregunta 2 - 6 puntos

Considere el siguiente programa escrito en pseudocódigo:

```
int foo = 2
int bar = 3


proc B(proc wee, int bar)
  if bar > 1
    B(wee,bar-1)
  wee(bar)
  print(bar)

proc A(int baz, int foo)
  proc fudge(int qux)
    bar := foo + bar + qux

  int bar = baz
  B(fudge,foo)
  print(bar)

main
  A(foo,2)
  print(bar)
end
```

Ejecute el programa aplicando las reglas de alcance y construcción de clausuras para cada uno de los casos especificados en la siguiente tabla. Escriba en la columna **Salida del Programa** los resultados emitidos durante cada corrida.

Alcance	**Asociación**	**Salida del Programa**
Estático	Profunda (*Deep*)	1 2 2 10
Dinámico	Profunda (*Deep*)	1 5 9 3
Dinámico	Superficial (*Shallow*)	4 6 2 3

Pregunta 3 - 7 puntos

Para manejar el problema de que el programador omita inicializar las variables, el libro de texto considera, entre otras alternativas, el uso de la estrategia "definite assignment". Los lenguajes que implantan esta verificación estática, como Java, intentan determinar si una variable particular está inicializada correctamente a través de *todos* los caminos de control posibles.
Suponga que Ud. está usando Haskell para implantar un lenguaje que incluirá esta verificación estática, que el compilador ya analizó sintácticamente el programa de entrada, y ha construido un árbol abstracto intermedio aprovechando los tipos de datos

```
data Stmt = Assign String Expr
          | Sequence [Stmt]
          | Cond Expr Stmt Stmt
          | Loop Expr Stmt

data Expr = ... -- no necesita definirlo
```

Así, el tipo `Stmt` representa las instrucciones: `Assign` representa la asignación a variables, `Sequence` representa la secuenciación de instrucciones, `Cond` corresponde al condicional `if/then/else`, y `Loop` corresponde a una iteración `while`.
El tipo `Expr` representa las expresiones, sean aritméticas o booleanas. Los detalles del tipo `Expr` no son relevantes, porque puede suponer que el compilador ya hizo las verificaciones de contexto necesarias para asegurar que todos los símbolos en uso han sido declarados y que los usos son cónsonos con el sistema de tipos. Es más, suponga que dispone de la función

```
occurs :: String -> Expr -> Bool
```

que permite determinar si un identificador de variable es utilizado en una expresión.
Con esta infraestructura, Ud. sólo debe implantar la función

```
initialized :: String -> Stmt -> Status
```

que determina si el identificador de variable correspondiente al primer argumento está inicializado de manera "definitiva" en la instrucción correspondiente al segundo argumento. El resultado será del tipo

```
data Status = Yes | No | Meh
```

donde `Yes` significa que la variable fue bien inicializada, `No` significa que la variable fue utilizada sin estar "definitivamente" inicializada, y `Meh` significa que la variable no fue inicializada pero tampoco se intentó usar su valor.
Nota: puede usar cualquier función del *Preludio* y escribir funciones auxiliares si le conviene.

```
initialized v (Assign l e) =
  if (occurs v e) then No
                  else if (v == l) then Yes else Meh

initialized v (Sequence is) =
  foldl f Meh (map (initialized v) is)
    where f Yes _ = Yes
          f No  _ = No
          f Meh r = r

initialized v (Cond b i0 i1) =
  if (occurs v b) then No
                  else f (initialized v i0) (initialized v i1)
    where f Yes Yes = Yes
          f No  _   = No
          f _   No  = No
          f _   _   = Meh

initialized v (Loop b i) =
  if (occurs v b) then No
                  else f (initialized v i)
    where f No = No
          f _  = Meh
```